Escuela Superior de Ingeniería

Programación en Internet

Grado en Ingeniería Informática

Invocación de un servicio web REST desde un cliente web y jQuery

Autores:
Javier Montes Cumbrera y Salvador Carmona Román

Supervisores:
Juan Boubeta Puig y Guadalupe Ortiz Bellot

Cádiz, 6 de mayo de 2015

Índice general

Índice de figuras

1. Invocación desde el cliente Web

En esta sección aprenderemos a cómo invocar el servicio que creamos en nuestro primer tutorial sobre la creación de un servicio Web REST y su despliegue en Tomcat.

1.1. Software necesario

Para poder realizar las peticiones tendremos que tener instalado algún navegador Web y algún complemento que nos permita la invocación del servicio indicando el tipo de petición y parámetros adicionales. En nuestro caso vamos a utilizar el siguiente software:

- Google Chrome [1].
- Simple REST Client para Google Chrome [2].

1.2. Explicación inicial de parámetros

En este apartado vamos a explicar cómo se forma la URL para invocar al servicio Web y los distintos parámetros que componen la petición.

1.2.1. URL del servicio

Para poder invocar las diferentes funcionalidades que hemos implementado en nuestro servicio Web tenemos que saber cómo formar las distintas URL. De manera general podemos distinguir varias partes en una llamada a un servicio web, estas serían:

- IP: Dirección IP del servidor, en nuestro caso sería *localhost*.
- Puerto: Puerto por el cual se puede acceder al servidor que alberga nuestro servicio Web, en nuestro caso se utilizará el puerto por defecto de Tomcat que es el 8080.
- Servicio: Este parámetro hace referencia al nombre que le dimos a nuestro servicio Web, en nuestro caso sería *WoERest*.

- *Pattern*: Cuando vimos la creación del fichero web.xml en nuestro anterior tutorial vimos que en la etiqueta *<url-pattern>* establecimos que permitiríamos el acceso a través de cualquier URL, por lo que este parámetro no existe para nosotros. Si al editar el fichero web.xml hubiésemos puesto otra cosa que no fuera /*, ahora mismo formaría parte de nuestra URL.
- *Path*: Parámetro @Path especificado en la clase y en los métodos de la clase. Para acceder a un método dentro de una clase tendremos que ir encadenando los diferentes *Path* hasta llegar al método que queramos utilizar.
- Parámetro: Estos serían los parámetros que queremos pasar a algún método de nuestra clase; podría no existir en alguno de los casos.

Después de ver todas las variables que puede tener nuestra URL, esta sería similar a la que podemos encontrar en el Listado 1.1.

Listado 1.1: Ejemplo de una URL para llamar a un servicio Web

```
1 http://IP:Puerto/Servicio/Path(clase)/Path(metodo)/Parametro
```

1.3. Probando el método GET

Para probar que nuestro servicio Web funciona correctamente primero ejecutaremos el método que nos devolvía la lista entera de miembros del comité en formato *JSON*. Una vez tengamos abierto el complemento Simple REST Client para Google Chrome, introduciremos la URL para llamar a nuestro servicio. En nuestro caso, la URL será *http://localhost:8080/WoERest/Rest/allMembers.*

En dicho complemento también tendremos que seleccionar que la llamada será de tipo *GET* y una vez pulsemos en *Send* tendríamos que poder ver algo parecido a la Figura 1.1.

1.4. Probando el método PUT

Para probar el método *PUT* tendremos que cambiar el tipo de petición en nuestro complemento, esta vez marcaremos la opción *PUT* y sustituiremos la URL anterior por *http://localhost:8080/WoERest/Rest/update/MarcoAiello.* Esto no es suficiente, ya que el método *PUT* se encarga de actualizar uno de los registros que tenemos en nuestra base de datos, por lo que también tendremos que enviar los nuevos datos. Para conseguir esto tenemos que incluir en el espacio *headers* que estamos enviando información en formato *JSON* con la línea del Listado 1.2.

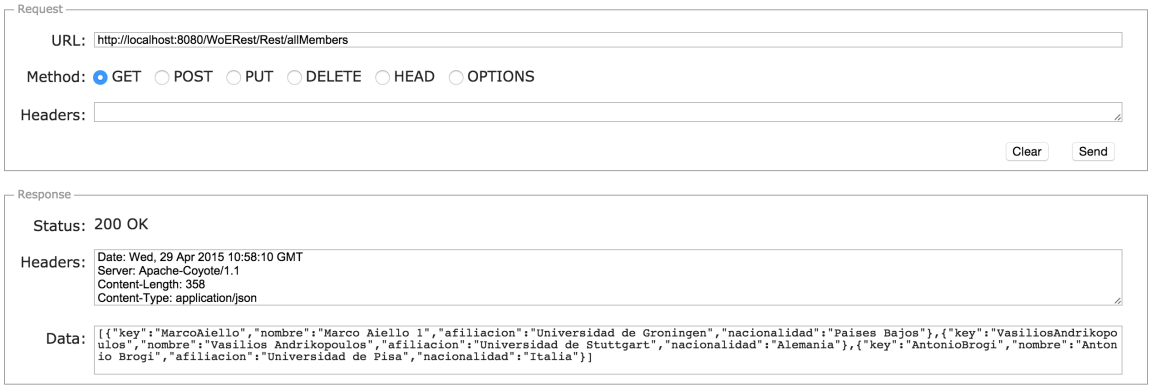

Figura 1.1.: Resultado de llamar al método *allMembers* del servicio Web.

Listado 1.2: Header para notificar que vamos a enviar datos en formato JSON

```
1 Content-Type: application/json
```

También tenemos que añadir en el campo *data* las líneas que podemos ver en el Listado 1.3 que serán los datos con los que nuestro servicio modifique el registro existente. Una vez rellenado todos los campos pulsamos en *Send* y, si todo ha funcionado correctamente, la respuesta del servidor será la que podemos ver en la Figura 1.2.

Listado 1.3: Datos en formato JSON

```
1 {"key":"MarcoAiello", "nombre":"Marco Aiello", "afiliacion":"Prueba
    Update", "nacionalidad":"Paises Bajos"}
```

1.5. Probando el método DELETE

Para probar el método *DELETE* tendremos que seguir los pasos anteriores y cambiar la URL por *http://localhost:8080/WoERest/Rest/delete/MarcoAiello*, de esta forma borraremos todos los datos relacionados con Miguel Angel de nuestra base de datos. Como podemos ver en la Figura 1.3, en este caso no es necesario enviar ninguna información adicional en los campos *headers* y *data*, simplemente hemos cambiado el tipo de petición a *DELETE*.

1.6. Probando el método POST

El último tipo de método que vamos a probar es el de tipo *POST* en el que incluiremos un nuevo miembro a nuestra base de datos. Para ello nos serviremos

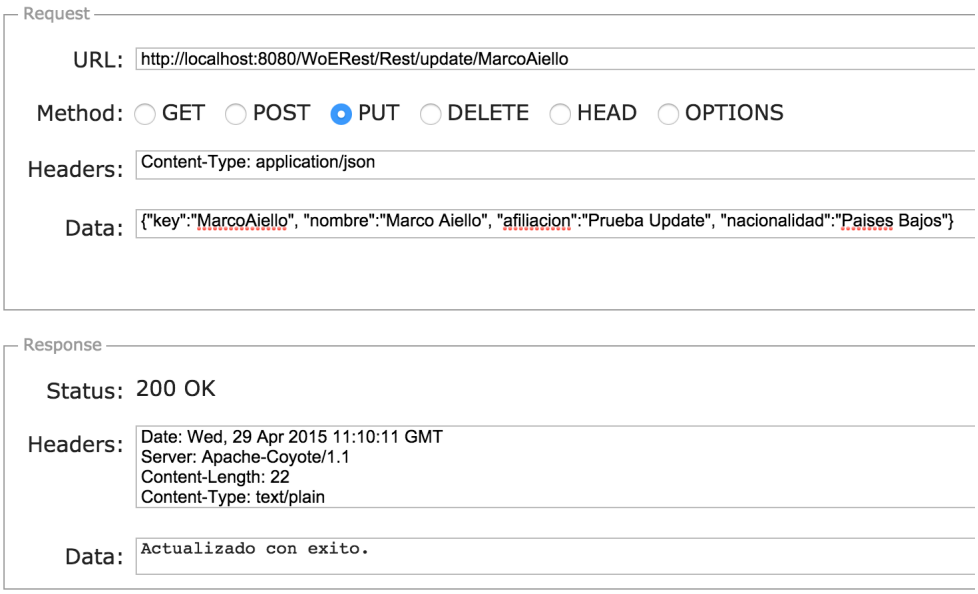

Figura 1.2.: Resultado de llamar al método *update* del servicio Web.

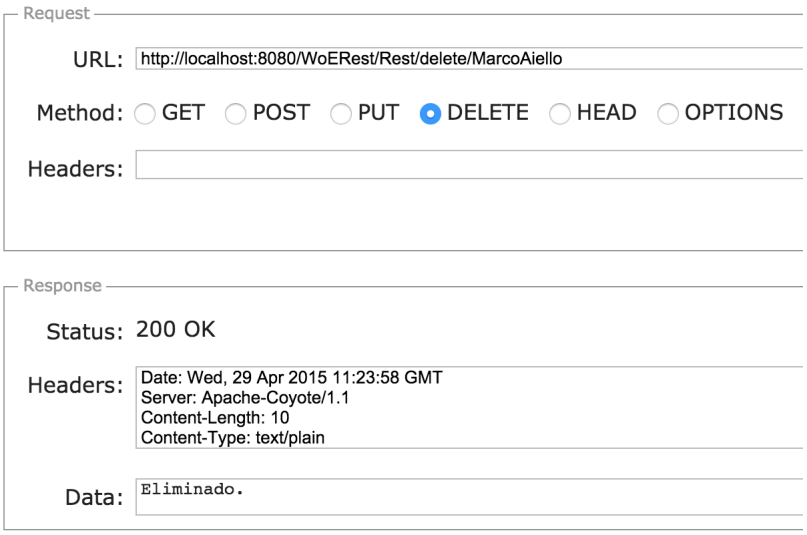

Figura 1.3.: Resultado de llamar al método *delete* del servicio Web.

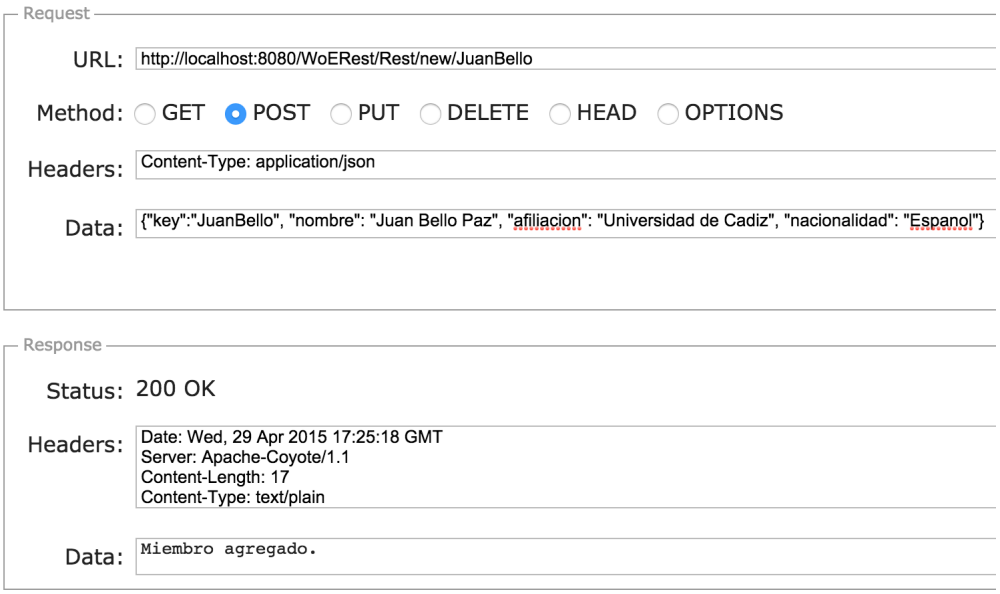

Figura 1.4.: Resultado de llamar al método *new* del servicio Web.

de la URL *http://localhost:8080/WoERest/Rest/new/JuanBello*. Al igual que vimos en la Sección 1.4 tenemos que incluir datos en el *header* (véase Listado 1.2) y en el apartado *data* (véase Listado 1.4). Una vez que rellenemos los campos y pulsemos en *Send* podremos ver cómo se ha introducido correctamente el nuevo miembro dentro de la base de datos (véase Figura 1.4).

Listado 1.4: Datos en formato JSON para añadir un nuevo miembro

```
1 {"key":"JuanBello", "nombre": "Juan Bello Paz", "afiliacion": "
    Universidad de Cadiz", "nacionalidad": "Espanol"}
```

2. Invocación desde jQuery

Ya hemos invocado los diferentes métodos que tiene nuestro servicio desde un cliente *REST*, pero no es la única forma de hacerlo. Otra de las formas para acceder a nuestro servicio, que de hecho es una de las más comunes, es a través de peticiones *jQuery* [3]. Para explicar cómo funcionan estas llamadas se detallaran unas cuantas funciones que ejecutan ciertos métodos que podemos encontrar en nuestro servicio.

2.1. Conocimientos previos

Durante todo el tutorial utilizaremos dos caracteres especiales para poder hacer referencia a las diferentes etiquetas que tengamos en nuestro HTML donde mostraremos los cambios (véase Anexo A). Estos caracteres son:

- $: Este carácter especial es una función de *jQuery* que busca todos los nodos (elementos) que concuerden con la expresión que escribamos en su interior.
- #: Este carácter sirve para indicar un id que se encuentre dentro del HTML que estemos utilizando.

Estos dos caracteres pueden combinarse para buscar dentro del HTML un elemento concreto, ya sea para obtener su valor o para cambiarlo, entre otras muchas opciones.

2.2. Función getComite

Esta función llamara al primero de los métodos que implementamos en nuestro servicio Web. En la petición *AJAX* en *jQuery* que tenemos en el Listado 2.1 podemos ver varios parámetros diferentes que se determinan de las siguiente forma:

- *type*: Este parámetro determina el tipo de petición que vamos a realizar. En nuestro caso, al querer obtener un listado completo, es de tipo *GET*.
- *url*: Como cabe esperar, este parámetro establece la URL de acceso al servicio Web. En nuestro caso es *http://localhost:8080/WoERest/Rest/allMembers*.
- *success*: Determina qué hacer con la respuesta del servidor cuando sea satisfactoria. En nuestra función creamos una lista con los diferente miembros que tenemos introducidos en nuestra base de datos.

- *error*: Si la respuesta del servidor no es satisfactoria, se puede decir que ha ocurrido un error. Para poder tratar este error nuestra función mostrará una ventana emergente notificando el fallo de la petición.

Listado 2.1: Función getComite

```
1  function getComite(){
2     $.ajax({
3        type: "GET",
4        url: "http://localhost:8080/WoERest/Rest/allMembers",
5        dataType:"json",
6        crossDomain: true,
7        success: function(data){
8           var html = "<ul>";
9           $.each(data, function(posicion, miembro){
10             html += '<li>' + miembro.nombre + ', ' + miembro.afiliacion
                   + ', ' + miembro.nacionalidad + '</li>';
11          });
12       html += "</ul>";
13       $("#contenido").html(html);
14       },
15       error:function(res){
16          $("#contenido").html("ERROR: "+ res.statusText);
17       }
18    });
19 }
```

2.3. Función getMiembro

Al igual que la función anterior, esta es de tipo *GET* pero con la diferencia de que solo recupera un miembro del comité indicando su clave a través de la URL. Al obtener un solo registro, el cuerpo de success será mucho más pequeño. Como podemos ver en la línea 2 del Listado 2.2, esta vez hemos decidido obtener como parámetro de un formulario Web la clave para obtener al miembro. Esto lo conseguimos indicando el id que hemos puesto en el campo a rellenar.

Listado 2.2: Función getMiembro

```
1  function getMiembro(){
2     var miembro = $("#memberKey").val();
3     $.ajax({
4        type: "GET",
5        url: "http://localhost:8080/WoERest/Rest/member/" + miembro,
6        success: function(data){
7           $("#contenido").html(data);
8        },
```

```
9         error:function(res){
10            alert("ERROR: " + res.statusText);
11        }
12    });
13 }
```

2.4. Función nuevoMiembro

La función nuevoMiembro, como su propio nombre indica, creará un nuevo miembro a partir de unos valores adquiridos a través de unas variables y realiza una petición *POST* pasándole como dato de tipo *JSON* toda la información necesaria recogida desde un formulario web. En el Listado 2.3 podemos ver 3 parámetros nuevos que no hemos visto en los casos anteriores, estos son:

- *contentType*: este parámetro determina el tipo de datos que vamos a enviar al servidor. En nuestro caso al tratarse de datos en formato *JSON* tendremos que notificarlo añadiendo «*application/json*».
- *dataType*: aunque pueda extrañar al principio, este parámetro notifica qué tipo de respuesta estamos esperando del servidor una vez que nuestra petición se resuelva satisfactoriamente. En nuestro caso estamos esperando solamente un texto informativo en formato plano, por lo que tenemos que notificarlo indicando «*text*».
- *data*: como bien indica su nombre aquí podemos encontrar los datos que queremos enviar a nuestro servicio Web. Como queremos enviar datos de tipo *JSON*, hemos utilizado la función JSON.stringify que convierte un string a formato *JSON*.

Listado 2.3: Función nuevoMiembro

```
1  function nuevoMiembro(){
2      var member = $("#memberKey").val();
3      var memberName = $("#memberName").val();
4      var memberAfil = $("#memberAfil").val();
5      var memberNation = $("#memberNation").val();
6      $.ajax({
7          type: "POST",
8          url: "http://localhost:8080/WoERest/Rest/new/" + member,
9          contentType: "application/json",
10         dataType: "text",
11         data: JSON.stringify({"key":member, "nombre": memberName, "
               afiliacion": memberAfil, "nacionalidad": memberNation}),
12         success: function(data){
```

```
13            $("#contenido").html(data);
14        },
15        error: function(res){
16            alert("ERROR "+  res.statusText);
17        }
18    });
19 }
```

2.5. Función eliminarMiembro

Esta función es muy sencilla ya que lo único que haremos será eliminar un miembro de los disponibles en nuestra base de datos. Como podemos ver en el Listado 2.4, no se diferencia mucho al descrito en la Sección 2.3, lo único que hemos cambiado es la URL de la petición indicando que queremos llamar al método delete.

Listado 2.4: Función eliminarMiembro

```
1  function eliminarMiembro(){
2      var miembro = $("#memberKey").val();
3      $.ajax({
4          type: "DELETE",
5          url: "http://localhost:8080/WoERest/Rest/delete/" + miembro,
6          dataType: "text",
7          success: function(data){
8              $("#contenido").html(data);
9          },
10         error: function(res){
11             alert("ERROR "+  res.statusText);
12         }
13     });
14 }
```

2.6. Función actualizarMiembro

La última función *jQuery* que vamos a utilizar es la encargada de actualizar un miembro del comité. Como en los casos anteriores, tendremos que añadir la clave del miembro en la URL y añadir como datos toda la información del miembro mediante datos de tipo *JSON*, por lo que volveremos a ayudarnos de la función JSON.stringify (véase Listado 2.5).

Listado 2.5: Función actualizarMiembro

```
1  function actualizarMiembro(){
2      var memberKey = $("#memberKey").val();
```

```
3     var memberName = $("#memberName").val();
4     var memberAfil = $("#memberAfil").val();
5     var memberNation = $("#memberNation").val();
6     $.ajax({
7        type: "PUT",
8        url: "http://localhost:8080/WoERest/Rest/update/" + memberKey,
9        contentType: "application/json",
10       dataType: "text",
11       data: JSON.stringify({"key":memberKey, "nombre": memberName, "
             afiliacion": memberAfil, "nacionalidad": memberNation}),
12       success: function(data){
13          $("#contenido").html(data);
14       },
15       error:  function(res){
16          alert("ERROR "+  res.statusText);
17       }
18    });
19 }
```

Bibliografía

[1] Google: Google Chrome (2015), https://www.google.es/chrome/browser/desktop/index.html

[2] Google: Simple REST Client (2015), https://chrome.google.com/webstore/detail/simple-rest-client/fhjcajmcbmldlhcimfajhfbgofnpcjmb

[3] jQuery: jQuery (2015), https://jquery.com

A. HTML

En este anexo se muestra el código HTML necesario para la comprobación del funcionamiento de las funciones *jQuery*.

```
1  <!DOCTYPE html>
2  <html lang="es">
3     <head>
4        <meta charset="utf-8">
5        <meta name="description" content="Pagina web de tutorial">
6        <meta name="viewport" content="width=device-width, initial-scale
             =1">
7        <title>Tutorial</title>
8        <script src="js/jquery-2.1.1.js"></script>
9        <script src="js/wsinvocation.js"></script>
10    </head>
11       <section>
12          <div><p id="contenido"></p></div>
13             KeyMember<input type="text" id="memberKey"><br>
14             Nombre:<input type="text" id="memberName"><br>
15             Afiliacion: <input type="text" id="memberAfil"><br>
16             Nacionalidad: <input type="text" id="memberNation"><br>
17             <br>
18             <button onClick="getComite();">getComite</button><br>
19             <button onClick="getMiembro();">getMiembro</button><br>
20             <button onClick="nuevoMiembro();">nuevoMiembro</button><br>
21             <button onClick="eliminarMiembro();">eliminarMiembro</
                   button><br>
22             <button onClick="actualizarMiembro();">actualizarMiembro</
                   button><br>
23       </section>
24    </body>
25 </html>
```

B. Funciones jQuery

En este anexo se muestran todas las funciones *jQuery* explicadas en este tutorial.

```
1  function getComite(){
2     $.ajax({
3        type: "GET",
4        url: "http://localhost:8080/WoERest/Rest/allMembers",
5        dataType:"json",
6        crossDomain: true,
7        success: function(data){
8           var html = "<ul>";
9           $.each(data, function(posicion, miembro){
10             html += '<li>' + miembro.nombre + ', ' + miembro.afiliacion
                   + ', ' + miembro.nacionalidad + '</li>';
11          });
12       html += "</ul>";
13       $("#contenido").html(html);
14       },
15       error:function(res){
16          $("#contenido").html("ERROR: "+ res.statusText);
17       }
18    });
19 }
20
21 function getMiembro(){
22    var miembro = $("#memberKey").val();
23    $.ajax({
24       type: "GET",
25       url: "http://localhost:8080/WoERest/Rest/member/" + miembro,
26       success: function(data){
27          $("#contenido").html(data);
28       },
29       error:function(res){
30          alert("ERROR: " + res.statusText);
31       }
32    });
33 }
34
35 function nuevoMiembro(){
36    var member = $("#memberKey").val();
37    var memberName = $("#memberName").val();
38    var memberAfil = $("#memberAfil").val();
```

```
39      var memberNation = $("#memberNation").val();
40      $.ajax({
41          type: "POST",
42          url: "http://localhost:8080/WoERest/Rest/new/" + member,
43          contentType: "application/json",
44          dataType: "text",
45          data: JSON.stringify({"key":member, "nombre": memberName, "
                afiliacion": memberAfil, "nacionalidad": memberNation}),
46          success: function(data){
47              $("#contenido").html(data);
48          },
49          error: function(res){
50              alert("ERROR "+  res.statusText);
51          }
52      });
53  }
54
55  function actualizarMiembro(){
56      var memberKey = $("#memberKey").val();
57      var memberName = $("#memberName").val();
58      var memberAfil = $("#memberAfil").val();
59      var memberNation = $("#memberNation").val();
60      $.ajax({
61          type: "PUT",
62          url: "http://localhost:8080/WoERest/Rest/update/" + memberKey,
63          contentType: "application/json",
64          dataType: "text",
65          data: JSON.stringify({"key":memberKey, "nombre": memberName, "
                afiliacion": memberAfil, "nacionalidad": memberNation}),
66          success: function(data){
67              $("#contenido").html(data);
68          },
69          error:  function(res){
70              alert("ERROR "+  res.statusText);
71          }
72      });
73  }
74
75  function eliminarMiembro(){
76      var miembro = $("#memberKey").val();
77      $.ajax({
78          type: "DELETE",
79          url: "http://localhost:8080/WoERest/Rest/delete/" + miembro,
80          dataType: "text",
81          success: function(data){
82              $("#contenido").html(data);
83          },
84          error: function(res){
85              alert("ERROR "+  res.statusText);
```

```
86              }
87          });
88      }
```